7 Arvores Binárias

" Testes experimentais de programas podem, quando muito mostrar a presença de erros, mas nunca podem mostrar a sua ausência "

- Niklaus Wirth -

Sumário:

7.1- Introdução

Nos capítulos anteriores estudamos as estruturas de dados lineares. Essas estruturas não são adequadas para resolver inumeras aplicações que necessitam de uma organização de forma hierarquica. As árvores são uma das estruturas mais importântes da área de computação, com inúmeras aplicações na vida real, cujos algoritmos de manipulação são simples e eficientes.

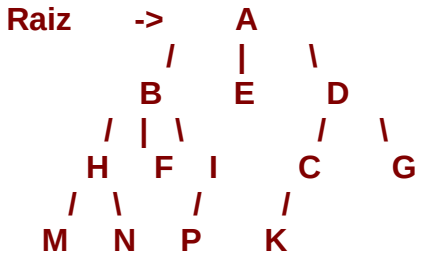
Intuitivamente uma árvore representa uma organização de informação, em que os dados têm uma relação de hiéraquia. Um dos exemplos mais conhecidos são árvores genealógicas que indicam os antepassados de um ser vivo e, o sistema de ficheiros de um sistema operativo.

Neste capítulo serão apresentados as definições básicas e os algoritmos que as manipulam. Ressalta-se que o estudo das árvores admite um tratamento puramente matemático, sem preocupação com a sua finalidade. Mas esse enfoque é completamente diferente ao que vamos apresentar nestas notas.

7.2- Definições, Terminologia e Propriedades

Uma **árvore genérica** é um conjunto finito de elementos denominados por átomos tais que: T é um conjunto não vazio constituído por um átomo especial denominado raiz e n ≥ 0 conjuntos disjuntos $T_1,T_2,\ldots,T_n$, que são subárvores da raiz. Essas subárvores também são árvores genéricas.

Mostramos em seguida uma figura que representa uma árvore genérica



Existem algumas convensões e terminologias que têm sido adoptadas universalmente em relação a este tema. Nestas notas, representaremos as àrvores de " cabeça para baixo", ou seja à raiz no topo e as subárvores em baixo.

Grau de um átomo é determinada pelo número de subárvores não vazias desse átomo. Por exemplo, para a árvore anterior, os átomos A, F, D, C têm graus três, zero, dois e um respectivamente.

Folha ou átomo terminal é um átomo de grau zero. Os restantes átomos são denominados por átomos internos ou átomos não terminais. O conjunto de folhas da árvore anterior é descrito por { M,N,F,P,E,K,G}.

Filhos de um átomo são as raizes de suas subárvores. Desse modo os filhos do átomo A são os átomos B, E e D. Também diz-se que A é **pai** de B, E e D. Nessa relação de parentesco, diz-se que os filhos do mesmo pai são **irmãos**. Os átomos E e D são irmãos porque são filhos de B. Obviamente poderiamos definir a noções de **primo**, **tio**, e por aí adiante.

Nível de um átomo é um número natural que satisfaz a seguinte relação de recorrência: A raiz de uma árvore possui nível zero; Se um átomo possui nível n os seus filhos possuem nivel n+1.

Profundidade ou **altura** de uma árvore é determinada pelo nível máximo de um átomo dessa árvore. Para a árvore do exemplo anterior os àtomos A,B,F e k têm níveis 0,1,2 e 3 respectivamente. Então essa árvore tem profundidade 3.

Caminho entre dois átomos, consiste numa sequência de átomos distintos entre entre si. Para a árvore da figura anterior, o caminho entre os átomos A e N é formado pela sequência : A, B, H, N.

Comprimento de um caminho é determinado pelo número de níveis entre o átomo origem para o átomo destino. O comprimento do caminho de A a N é três.

Diz-se que uma **árvore** está **ordenada** quando os filhos de qualquer átomo estão ordenados em ordem crescente ou decrescente da esquerda para à direita.

Grau de uma árvore é determinado pelo maior valor do grau dos seus átomos. A árvore anterior possui grau 3.

Ascendente de um átomo N é um átomo A, se existe um caminho de A a N diferente de zero.

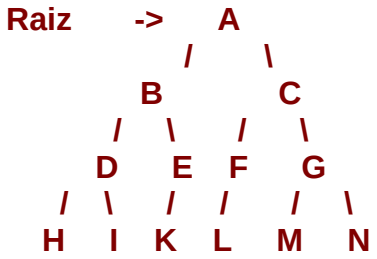
Floresta é um conjunto de zero ou mais árvores disjuntas.

7.3- Árvores Binárias

Conforme mencionamos, as árvores constituem as estruturas não lineares com a maior aplicação na computação. Dentre elas, as árvores binárias são as que mais se destacam.

Uma árvore binária **(binary tree)** é um conjunto finito T de átomos tais que: T é vazio ou existe um átomo especial chamado raiz e os restantes átomos estão particionadas em duas subárvores binárias disjuntas T_e e T_d, denominadas por subárvore à esquerda e à direita da raiz, que também são árvores binárias.

Apresentamos em seguida uma figura que representa uma árvore binária.



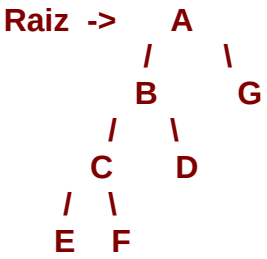
As principais diferenças entre as árvores binárias e as árvores genéricas residem nos seguintes factos: Uma árvore binária é um conjunto vazio enquanto uma árvore genérica não é; Nas árvores genéricas, não se distingue à subárvore à esquerda da subárvore à direita. Em termos mais precisos, dadas duas árvores com a mesma estrutura e a mesma raiz, se permutarmos o conteúdo dos átomos das subárvores da raiz, temos árvores genéricas iguais, mas árvores binárias diferentes.

Existem algumas propriedades importantes para as árvores binárias. Citaremos apenas a mais imporânte, mas deixaremos a sua demonstração para o leitor que pretender aprofundar este assunto. Essa demonstração é bastante simples e basea-se no método de indução finita.

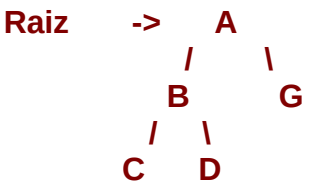
Teorema 1: *O número máximo de átomo numa árvore binária no nível i é* 2^i *, para qualquer* $i \geq 0$.

Antes de continuarmos com o tema, devemos salientar que são validas para as árvores binárias, as definições e a notação utilizada para as árvores genéricas. Iremos em seguida, introduzir mais alguns conceitos importantes sobre árvores binárias especiais.

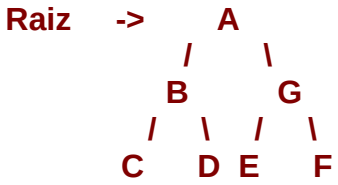
Uma árvore binária **é estritamente binária** se os seus átomos possuem zero ou dois filhos. Por exemplo:



Uma **árvore binária de profundidade h é quase completa** se todas suas folhas estão nos nível h-1 e h. Para além disso, o nível h-1 está totalmente preenchido e as folhas que estão no nível h encontram-se o mais a esquerda possível.



Uma **árvore binária completa** é uma árvore binária em que todas as folhas estão no mesmo nível.



Em função destes conceitos podemos enunciar mais um propriedade.

Teorema 2: *Uma árvore binária completa de profundidade h possui* $2^{h+1} - 1$ *átomos para qualquer* $h \geq 0$.

7.4- Tipo Abstracto de Dados

Um possível conjunto de operações para criar um tipo abstracto de dados árvore binária é descrito a seguir:

- Criar uma árvore vazia;
- Criar um átomo com a raiz;
- Verificar se a árvore está vazia;
- Inserir um elemento na árvore;
- Remover um elemento da árvore;
- Consultar um elemento na árvore.

7.5- Implementação Dinâmica

O armazenamento de árvores pode ser feito em organização sequencial ou dinâmica. As vantagens e desvantagens dessas implementações já foram estudadas. Mas, devido a natureza recursiva desta estrutura faremos apenas a implementação em organização dinâmica.

7.5.1- Códigos de Erro

```
#define NOT_FOUND      -1        // Item não existe
#define  OK             0        // Operação realizada com sucesso
#define  TREE_EMPTY     2        //  Arvore vazia
#define NO_SPACE        5       // Não há espaço de memória
```

7.5.2- Estrutura de Dados

```
typedef struct
 {
    int   chave;
    int   valor;
 }TInfo;

typedef struct Atomo
 {
   struct Atomo *fesq;
  TInfo   info;
  struct Atomo *fdir;
 }TAtomo;
```

Declaramos uma árvore binária em organização dinâmica com um ponteiro raiz que faz referencia ao primeiro átomo da árvore.

```
typedef struct
  {
      TAtomo * raiz;
   } TAB;

typedey  enum   {FALSE= 0, TRUE= 1} Boolean;
```

7.5.3- Implementação das Operações

Inicializar uma árvore binária consiste em associa-la à um ponteiro que fará referencia ao seu átomo raiz. O valor desse ponteiro será inicializado com um endereço nulo para indicar que essa árvore está vazia.

```
*/-------------------------------------------------------------------------------------------------
Objectivo: Inicializar  uma estrutura de dados do tipo árvore binária
Parâmetro Entrada: Árvore binária
Parâmetro de Saída:  Árvore binaria  com o ponteiro raiz actualizado.
------------------------------------------------------------------------------------------------- */
 void  iniciaArvoreBinaria (TAB *arvore)
{
   arvore-> raiz = NULL;
}
```

Inicializar um átomo como raiz, consiste em solicitar um átomo à memória e associa-lo ao ponteiro raiz. Preencher em seguida esse átomo com a informação e um ponteiro nulo nos campos filho esquerdo e direito.

```
*/-------------------------------------------------------------------------------------------------
Objectivo: Inicializar  uma estrutura de dados do tipo árvore binária
Parâmetro Entrada:  Árvore binária
Parâmetro de Saída: Árvore binaria Actualizada
Valor de Retorno: Correspondente código de erro ( NO_SPACE ou OK)
------------------------------------------------------------------------------------------------- */
```

```
int  iniciaAtomoRaiz (TAB  *arvore, TInfo  x)
{
   arvore->raiz = (TAtomo *)malloc(sizeof(TAtomo));
   if (arvore->raiz == NULL)
      return NO_SPACE;
   arvore->raiz->info = x;
   arvore->raiz->fesq = NULL;
   arvore->raiz->fdir = NULL;
   return OK;
}
```

A operação para inserir um átomo como filho esquerdo, consiste em verificar em primeiro lugar se a árvore está vazia. Se essa condição for verdadeira criar um átomo com a informação que se pretende inserir. No caso contrário, localizar o átomo-pai. Se o átomo-pai não for encontrado devolver o correspondente código de erro. Se for encontrado verificar se esse átomo já tem um filho esquerdo. Se essa condição for verdadeira, devolver o correspondente código de erro. No caso contrário inserir um novo átomo com a informação. A inserção é muito simples, consiste em ligar o novo átomo ao seu pai e considera-lo como folha.

```
*/-------------------------------------------------------------------------------------------------
Objectivo: Inserir um novo átomo como descendente de uma chave
Parâmetro Entrada: Árvore binária,  informação inserir e chave do átomo pai
Parâmetro de Saída: Árvore binaria actualizada
Valor de Retorno: Correspondente código de erro.
------------------------------------------------------------------------------------------------- */
int  inserirFilhoEsquerdo (TAB *arvore, TInfo x, int chavepai)
{
    if ( VaziaArvoreBinaria (&arvore))
       return IniciaAtomoRaiz (arvore,x);
    else {
           TAtomo * pai = localizar (&arvore,chavepai);
            if ( pai == NULL)
               return NOT_FOUND;
            if ( pai->fesq != NULL)
               return  DESCENDING_FOUND;
            TAtomo * pfilho  = (TAtomo *)malloc(sizeof(TAtomo));
             if ( pfilho == NULL)
               return NO_SPACE;
             pfilho->fesq = NULL;
             pfilho->fdir = NULL;
             pfilho->info = x;
             pai->fesq = pfilho;
             return OK;
         }
}
```

A operação para remover um átomo numa árvore binária envolve dois casos: Se o átomo for uma folha o processo é mais simples e consiste em cortar a sua

ligação com o átomo pai. Mas se o átomo for interno esse processo é muito complexo porque necessitamos de reorganizar a árvore. Nestas notas mostraremos apenas como remover uma folha de uma árvore binária. Esta operação consiste em verificar em primeiro lugar se a árvore está vazia. Se essa condição for verdadeira devolver o correspondente código de erro. No caso contrário, localizar o átomo pai. Se o átomo pai não for localizado devolver o correspondente código de erro. No caso contrário cortar a ligação se for uma folha.

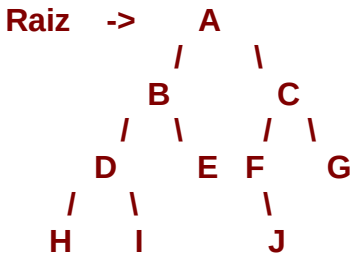
```
*/-------------------------------------------------------------------------------------------------
Objectivo: Remover uma folha dada a chave do átomo-pai
Parâmetro Entrada:  Árvore binária, chave a remover, chave do átomo pai
Parâmetro de Saída: Árvore binária actualizada, conteudo da informação
Valor de Retorno: Código de erro.
------------------------------------------------------------------------------------------------- */
int  removerFolha (TAB  *arvore, TInfo * x, int chavepai, int chave)
{
    if ( VaziaArvoreBinaria (&arvore))
       return TREE_EMPTY;
    TAtomo  * pai = localizar(&arvore,chavepai);
    if ( pai == NULL)
        return NOT_FOUND_FATHER;
    if ( pai->fesq == NULL && pai->fdir == NULL)
        return NOT_DESCENDING;
    if ( pai->fesq != NULL  &&  pai->fesq-> info.chave == chave)
      {
         TAtomo  * pdel = pai->fesq;
         if (pdel->fesq = NULL  && pdel->fdir = NULL)
            {
              *x = pdel->info;
               pai->fesq = NULL;
               free(pdel);
               return OK;
             }
          return NOT_TERMINAL;
      }
    if (pai->fdir != NULL  &&  pai->fdir-> info.chave == chave)
      {
          TAtomo * pdel = pai->fdir;
           if (pdel->fesq = NULL  && pdel->fdir = NULL)
              {
                  *x = pdel->info;
                   pai->fesq = NULL;
                   free(pdel);
                   return OK;
              }
           return  NOT_TERMINAL;
      }
     return NOT_FOUND_TERMINAL;
}
```

A operação para consultar uma informação, consiste em percorrer a árvore a partir da raiz. Mas os algoritmos de busca que conhecemos irão passar por alguns átomos da árvore mais do que uma vez. Por exemplo, se a raiz possuir dois descendentes a consulta a cada um desses descendentes é feita pela raiz. Como consequência, os algoritmos tradicionais de busca, passarão pela raiz duas vezes. O nosso problema consiste em desenvolver um algoritmo que consulte cada átomo da árvore uma e apenas uma vez.

7.5.4- Percursos em Árvores Binárias

Os átomos de uma árvore são consultados para realizar uma determinada tarefa. Na ciência de Computação á realização dessa tarefa recebe o nome de **visita**.

Percorrer uma árvore binária, consiste em visitar os seus átomos uma e apenas uma vez. Esse percurso produz uma sequência linear de informações contidas na árvore. O conteúdo dessa sequência depende da ordem do percurso. Se considerarmos que E,V,D representam, percorrer à esquerda, visitar (aceder) a informação do átomo e percorrer à direita, para um determinado átomo teremos seis combinações possíveis: EVD, EDV, DEV, DVE, VED e VDE. Se adoptarmos o princípio de percorrer à esquerda antes de percorrer à direita, ficaremos com apenas três combinações: EVD, EDV e VED. Estas combinações têm nomes especiais na Ciência da Computação e serão objecto de estudo mas próximas linhas. Para efeitos didácticos, utilizaremos como exemplo, a seguinte árvore:



A operação de visita, será descrita pelo seguinte procedimento.

```
void  visitaAtomo (int  x)
{
   printf(" %c ", x);
}
```

Percorrer uma árvore binária em **pré-ordem,** consiste em: Visitar o átomo; Percorrer de forma recursiva à subárvore esquerda em pré-ordem; Percorrer de forma recursiva a subárvore à direita em pré-ordem.

Este percurso, gera a seguinte sequência de caracteres: A, B, D, H, I, E, C, F, J, G e é descrito pela seguinte função recursiva.

```
*/-------------------------------------------------------------------------------------------------
Objectivo: Percorrer a árvore em pre-ordem
Parâmetro de Entrada:  O endereço da raiz de uma árvore binária
------------------------------------------------------------------------------------------------- */
void preOrdemRecursivo (TAtomo  *raiz)
{
   if ( raiz != nulo)
      {
         visitaAtomo (raiz->info.valor);
         preOrdemRecursivo (raiz->fesq);
         preOrdemRecursivo (raiz->fdir);
       }
}
```

Vejamos a seguir a versão iterativa que utiliza uma estrutura de pilha, Supomos que cada elemento dessa pilha é do tipo caracter.

```
*/-------------------------------------------------------------------------------------------------
Objectivo: Percorrer a árvore em pre-ordem
Parâmetro de Entrada:  Endereço da raiz da árvore binária
------------------------------------------------------------------------------------------------- */
int  preOrdemIterativo (TAtomo *raiz)
{
  if (raiz == NULL)
     return TREE_EMPTY;
  TAtomo  * paux = raiz;
  TPilhaDinamic  *pilha;
  iniciaPilhaDinamica(&pilha);
  do  {
         while ( paux != NULL)
           {
              visitaAtomo (paux->info);
              int codret = empilhar (pilha,paux->info);
              if (codret == NO_SPACE)
                 {
                    destruirPilha (&pilha);
                    return NO_SPACE;
                  }
              paux = paux->fesq;
           }
           desempilhar (&pilha,paux->info);
           paux = paux->fdir;
       }
   while (pilha->topo != NULL || paux != NULL)
   return TRUE;
 }
```

Façamos uma simulação do algoritmo para a árvore que utilizamos como exemplo. Colocaremos para efeitos de notação a pilha na posição horizontal.

Inicialmente a raiz aponta para o átomo A. Como A não é nulo, então associamos ao endereço da raiz um ponteiro auxiliar e inicializamos uma pilha. Como o ponteiro auxiliar é diferente de nulo, acedemos a informação do átomo , colocamos o endereço de A na pilha, que por aspecto didácticos dizemos " empilhamos A" , em seguida, movimentamos o ponteiro auxiliar para o filho esquerdo. Nesse instante temos: Pilha (A), o ponteiro auxiliar aponta para o endereço de B. Como o endereço de B não é nulo, acedemos a informação de B, empilhamos B e o ponteiro auxiliar aponta para o endereço do filho esquerdo de B. Nesse instante temos: Pilha (A,B), o ponteiro auxiliar aponta para endereço de H. Como o endereço de H não é nulo, acedemos a informação desse átomo, empilhamos H e ponteiro auxiliar aponta para o endereço do filho esquerdo de H. Nesse instante temos: Pilha(A,B,D,H) e o ponteiro auxiliar aponta para um endereço nulo. Como o endereço é nulo, retiramos da pilha o endereço aponta para H e movimentamos o ponteiro auxiliar para o endereço do filho direito de H. Como a pilha não esta vazia, repetimos mais um laço. O endereço do filho direito de H é nulo, logo retiramos da pilha o endereço que aponta para o átomo D e movimentamos o ponteiro auxiliar para o endereço do filho direito de D. Como a pilha não está vazia e o ponteiro auxiliar é diferente de nulo, repetimos mais um laço. Acedemos a informação do átomo I, empilhamos o endereço desse átomo e movimentamos o ponteiro para o filho esquerdo de I. Nesse instante temos: Pilha(A,B,I), o ponteiro auxiliar aponta para endereço nulo. Continue...

Percorrer uma árvore binária em In-ordem consiste em: Percorrer de forma recursiva a subárvore à esquerda em in-ordem; Visitar o átomo; Percorrer de forma recursiva a subárvore à direita em in-ordem.

Este percurso, gera a seguinte sequencia: H D I B E A F J C G e é descrito pela seguinte função recursiva:

```
*/-------------------------------------------------------------------------------------------------
Objectivo: Percorrer a árvore em In-ordem
Parâmetro de Entrada: Endereço da raiz da árvore binária
------------------------------------------------------------------------------------------------- */
void  inOrdemRecursivo (TAtomo *raiz)
{
   if ( raiz != NULL)
     {
        iniOrdemRecursivo(raiz->fesq);
        visitaAtomo(raiz->info.valor);
         inOrdemRecursivo(raiz->fdir);
     }
 }
```

A versão iterativa é descrita pela seguinte função.

```
*/-------------------------------------------------------------------------------------------------
Objectivo: Percorrer a árvore em In-ordem
Parâmetro de Entrada: Endereço da raiz da árvore binária
------------------------------------------------------------------------------------------------- */
```

```
int  inOrdemIterativo (TAtomo  *raiz)
{
  if ( raiz == NULL)
     return TREE_EMPTY;
  TAtomo *paux = raiz;
  TPilhaDinamic  *pilha;
   iniciaPilhaDinamica (&pilha);
   do {
          while ( paux != NULL)
            {
               int codret = empilhar (&pilha,paux->info);
               if (codret == NO_SPACE)
                  {
                      destruirPilha (&pilha);
                     return NO_SPACE;
                  }
               paux = paux->fesq;
            }
          desempilhar (&pilha,paux->info);
          VisitaAtomo(paux->info.valor);
          paux = paux->fdir;
       }
   while (pilha->topo != NULL ||  paux != NULL)
    return OK;
  }
```

Percorrer uma árvore binária em pos-ordem consiste em: Percorrer de forma recursiva a subárvore à esquerda em pos-ordem; Percorrer de forma recursiva a subárvore à direita em pos-ordem; Visita à informação do átomo.

Este percurso, gera a seguinte sequência: H I D E B J F G C A e é descrito pelo seguinte função recursiva:

```
*/-------------------------------------------------------------------------------------------------------
Objectivo: Percorrer a árvore em pos-ordem
Parâmetro de Entrada: Endereço da raiz da árvore binária
------------------------------------------------------------------------------------------------------- */
void  posOrdemRecursivo(TAtomo *raiz)
{
   if ( raiz != NULL)
     {
        posOrdemrecursivo (raiz->fesq);
        posOrdemrecursivo (raiz->fdir);
        visitaAtomo (raiz->info.valor);
      }
 }
```

7.5.5- Outras Operações

Com base na definição recursiva de percursos, podemos fácilmente redigir novos operadores. Por exemplo, contar o número de átomos de uma árvore binária. Pegamos no algoritmo de percurso em pré-ordem e alteramos a função visitatomo de modo a produzir a seguinte função:

```
int contaAtomos (TAtomo *raiz)
{
  if (raiz == NULL)
     return 0;
  return 1+ contaAtomos (raiz->fesq) + contaAtomos (raiz->fdir);
}
```

Outro problema simples, baseado no mesmo princípio é produzir uma cópia de uma árvore binária.

```
TAtomo *copiaArvore (TAtomo *raiz )
{
 TAtomo *paux =  (TAtomo *) malloc(sizeof(TAtomo));
  if (paux != NULL)
     if (raiz != NULL)
       {
          paux->info =  raiz->info;
          paux->fesq = copiaArvore (raiz->fesq);
          paux->fdir =  copiaArvore (raiz->fdir);
       }
   return paux;
 }
```

Duas árvores binárias são equivalentes quando têm a mesma topologia. A expressão mesma topologia, que dizer que para cada ramo da primeira árvore corresponde a um ramo da segunda na mesma ordem. A utilização do algoritmo recursivo de percursos é uma excelente ferramenta para resolver este problema. O algoritmo iguais percorre duas árvores binárias em préordem, embora qualquer ordem poderia ser utilizada.

```
Boolean  iguais (TAtomo *raiz1, TAtomo *raiz2)
{
  if (raiz1 == NULL && raiz2 == NULL)
     return true;
  if ( raiz1 != NULL && raiz2 != NULL)
    if ( raiz1->info == raiz2->info)
      return iguais (raiz1->eprox,raiz2->eprox) &&
               iguais (raiz1->dprox,raiz2->dprox);
   else
       return false;
  else
     return false;
}
```

Aceder a todos os átomos de um determinado nível, consiste em associar ao percurso pré-ordem um índice, que é decrementado a medida que baixamos no nível na árvore. Quando atingirmos o nível desejado o índice contêm o valor zero e o átomo é acedido.

```
void visitaNivel  (TAtomo *raiz, int nivel)
{
  if ( raiz != NULL)
     if (nivel == 0)  visitaAtomo (raiz->info.valor);
     else  {
               visitaNivel (raiz->fesq, nivel-1);
               visitaNivel (raiz->fdir,nivel-1);
             }
}
```

Percorrer uma árvore no sentido de cima para baixa (Top-Down), nível por nível e da esquerda para a direita, fica como exercício para consolidar a matéria. Sugerimos a utilização do algoritmo anterior para resolver o problema.

Para calcular o elemento máximo, suponho que todos os átomos possuem conteúdo positivo, basta pegar no algoritmo de percurso em pré-ordem e alteramos a função visitatomo de modo a produzir o seguinte algoritmo:

```
int maximo (TAtomo *raiz)
{
 if (raiz == NULL)
    return 0;
 else  {
           int maxe = compara (raiz->info.valor, maximo(raiz->fesq));
           int maxd = compara (raiz->info.valor,maximo(raiz->fdir));
          return  compara (maxe,maxd);
        }
 }
```

A profundidade ou altura de uma árvore, é determinada pelo número máximo de seus níveis. Para implementar este operador, temos duas opções: Alterar o algoritmo do exercício anterior, ou alterar o algoritmo para percorrer uma árvore em pré-ordem. A primeira possibilidade fica como exercício.

```
int altura (TAtomo *raiz)
{
  int  alte, altd;
  if ( raiz == NULL )  return 0;
  else {
           alte ← 1+ altura(raiz->fesq);
           altd ← 1+ altura(raiz->fdir);
           if (alte > altd)   return alte + 1
           else  return altd+1
        {
{
```

A operação para construir uma árvore binária, depende da ordem em que são fornecidos os valores dos seus átomos. Essa ordem está relacionada com o percurso na árvore, isso quer dizer que a árvore gerada deverá ser percorrida nessa ordem. Tomemos sem perda da generalidade o percurso em pré-ordem. O primeiro valor lido, será armazenado na raiz. Isso quer dizer que devemos desenvolver uma função para criar uma árvore com um único átomo. Essa função é espressa pelo segmento de código.

```
Boolean constroiRaiz (TAtomo *raiz )
{
  if ( vaziaArvoreBinaria(&raiz) )
     if ( iniciaAtomoRaiz(&raiz,lerValor())== OK )
        return TRUE;
    else
       return FALSE;
  else
     return FALSE;
}
```

Se essa operação for bem sucedida, iniciamos o processo de construção da subárvore esquerda, colocando como filho esquerdo da raiz o endereço da raiz dessa subárvore. Terminado esse processo, vamos construir a subárvore direita, colocando como filho direito da raiz o endereço da raiz dessa subárvore.

Pela definição do método, podemos constactar que o processo de construção das subárvores pode ser implementado por um algoritmo recursivo.

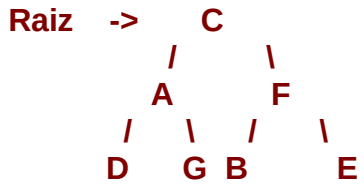
```
TAtomo *constroiSubArvore (TAtomo *raiz )
{
  int x = LerValor();
  if ( x == 0 )
     return raiz;
  TAtomo *paux =  (TAtomo *) malloc(sizeof(TAtomo));
  if (paux != NULL)
     {
         paux->info =  x;
         paux->fesq = NULL;
         paux->fdir  = NULL;
         raiz->fesq  = constroiSubArvore(paux);
         raiz->fdir   =  ConstroiSubArvore(paux);
      }
   return paux;
}
```

7.6- Reconstituição de Árvores Binárias

A reconstituição de uma árvore binária, pode ser feita se tivermos conhecimento de pelo menos dois percursos. Com os percursos: {pré-ordem, in-ordem} e {pos-ordem, in- ordem} fácilmente se reconstitui uma árvore binária, mas se tivermos os percursos {pré- ordem, pos-ordem} só poderemos reconstituir a correspondente árvore binária se cada átomo possuir exactamente dois filhos. A título de exemplo:

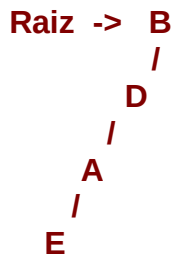
1- Dados os seguinte percursos:
Pré-ordem : C A D G F B E
In-Ordem : D A G C B F E

Façamos uma descrição do método de reconstituição. Percorrer à sequência em pré-ordem e analisar à posição desse caracter na sequência In-ordem. Iniciamos com o átomo C. C é a raiz da árvore. Mas, pela definição da sequência em In-ordem temos como subárvore à esquerda: D A G e árvore à direita : B F E. O caracter a seguir na sequência pré-ordem o átomo A. Como A está esquerda de C na sequência Ins-ordem então A é o seu filho esquerdo. Mas, pela definição de in-ordem, A tem duas subárvores: D como subárvore à esquerda e G como subárvore à direita. Mas essas subárvores são unitárias, logo, D e G são os filhos esquerdos e direitos de A. Continuando o processo chegaremos a seguinte árvore:



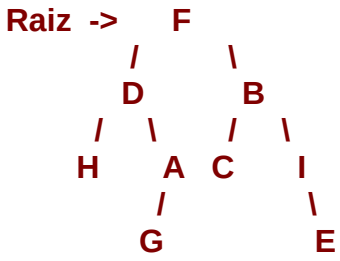
2- Dados os seguintes percursos:
In-ordem: E A D B
Pos-Ordem: E A D B

A estratégia para reconstruir essa árvore consistem em percorrer a sequência em pos-ordem no sentido inverso e analisar a posição desse caracter na sequência in-ordem. Iniciamos com o átomo B. B é a raiz a árvore. Mas pela sequência in-ordem essa árvore não possui sub-árvore à direita, logo o átomo a seguir da sequência pos-ordem é o filho esquerdo. Continuamos esse processo até a obtenção da seguinte árvore:



3 - Dados os seguintes percursos:
Pré-Ordem : F D H A G B C I E
In-Ordem : H D G A F C B I E

Estes percursos representam a seguinte árvore:

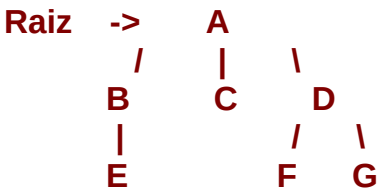


7.7- Transformação de Árvores Genéricas em Binárias

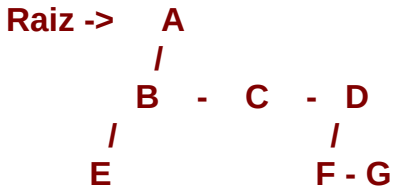
Se uma aplicação necessitar de estruturar os seus dados através de uma árvore n-ária, é sempre possível transformar essa representação para uma árvore binária equivalente. O objectivo dessa transformação é, para além da facilidade da representação da árvore binária, possibilitar a utilização dos algoritmos de manipulação. A correspondente árvore binária terá sempre o mesmo número de elementos da árvore original. O processo de transformação é descrito pelo seguinte método:

- Ligar todos os irmãos de cada átomo
- Apagar todas as ligações de um átomo com os seus descendentes, com excepção do primeiro filho, que é o filho mais a esquerda.

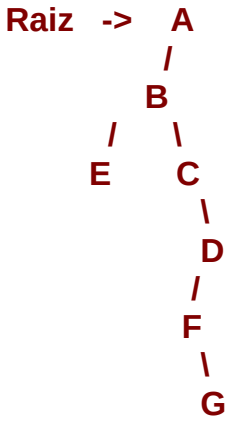
A título ilustrativo, veremos como converter a seguinte árvore genérica:



Iniciamos a conversação pela raiz. Solicitamos um átomo, inserimos o valor A e associamos a um ponteiro especial, para indicar a raiz da árvore. A raiz não tem irmãos, seus descendentes são os átomos B , C e D. Portanto o ponteiro irmão de A contêm o valor nulo e o ponteiro descendente aponta para o átomo B. Como C e D são irmãos de B, então esses átomos formam uma lista ligada através do seu ponteiro irmão. Por sua vez, B possui um descendentes E que formam uma lista ligada unitária cujo início é indicado pelo ponteiro descendente do átomo B. Esse processo continua até a representação da seguinte figura:



Contudo, essa árvore não tem um aspecto de uma árvore binária. Mas, cada átomo possui no máximo dois filhos. Se considerarmos descente como subárvore a esquerda , irmão como subárvore a direita e inclinarmos a árvore genérica, obtemos uma árvore binária:



7.8- Conversão de Florestas em Árvores Binárias

Para converter uma floresta numa árvore binária, devemos utilizar o mesmo método. Em primeiro lugar, tranformar as árvores genéricas dessa floresta em arvores binárias e em seguida ligar todas as árvores binárias pelo campo irmão de suas raizes.

7.9- Exercícios

7.9.1- Desenvolva uma função que recebe como parâmetro de entrada um ponteiro para a raiz de uma árvore binária e uma determinada informação. Inserir essa informação como filho esquerdo.

7.9.2- Generalize o exercício anterior de tal á inserir um átomo.

7.9.3- Desenvolva um procedimento recursivo que recebe como parâmetro de entrada um ponteiro para a raiz de uma árvore binária. Percorrer esse árvore nível por nível da esquerda para direita e de cima para baixo

7.9.4- Desenvolva uma função recursiva que recebe como parâmetro de entrada um ponteiro para a raiz de uma árvore binária. Contar o número de átomos que são folhas.

7.9.5- Desenvolva uma função recursiva que recebe como parâmetro de entrada dois ponteiros para as raízes de duas árvores binárias. Verificar se essas árvores são opostas, ou seja, se a subárvore à esquerda de uma é igual à subárvore a direita de outra e vice-versa.

7.9.6- Desenvolva uma função recursiva que recebe como parâmetro de entrada um ponteiro para a raiz de um árvore binária. Devolver como valor de retorno o conteúdo do átomo com o valor mínimo.

7.9.7- Desenvolva uma função recursiva que recebe entre outros parâmetros de entrada, um ponteiro para a raiz de uma árvore binária. Devolver como valor de retorno o número de átomos de descendentes que não são folhas.

7.9.8- Desenvolva uma função iterativa que recebe como parâmetro de entrada um ponteiro para a raíz de uma árvore binária. Verificar se essa árvore e estritamente binária, ou seja, todos os átomos têm dois filhos com a excepsão das folhas.

7.9.9- Desenvolva uma função iterativa que recebe como parâmetro de entrada um ponteiro para a raiz de uma árvore binária, uma determinada chave e uma informação. Inserir essa informação como descendente de um átomo identificado pela sua chave.

7.9.10- Desenvolva uma função iterativa que recebe como parâmetro de entrada um ponteiro para a raiz de uma arvore binária e uma determinada informação. Inserir essa informação como a folha mais a esquerda dessa árvore.

7.9.11- Desenvolva uma função iterativa que recebe como parâmetro de entrada um ponteiro para a raiz de uma árvore binária não vazia. Devolva o endereço do primeiro átomo da árvore percorrida em ordem simétrica (in-ordem).

7.9.12- Desenvolva uma função que recebe como parâmetro de entrada um ponteiro para uma árvore binária. Devolver como parâmetro de saída a correspondente àrvore binária com o campo pai preenchido. Suponha que esse caso foi inicializado com o valor NULL.

7.9.13- Reconstrua a árvore binária a partir dos percursos em pré-ordem e in-ordem:

Pré-ordem: N J I H G A K F B M L E C D
In-ordem: H I A G J F B K N L C E D M

7.9.14- Desenvolva uma função iterativa que recebe como parâmetro de entrada um ponteiro para a raiz de uma árvore binária. Devolver como valor de retorno o número de átomos que são folhas.

7.9.15- Desenvolva uma função recursiva que recebe entre outros parâmetros de entrada um ponteiro para a raiz de uma árvore binária a1 e um ponteiro para a raiz de uma árvore binária a2. Verificar se uma árvore binária está contida numa outra árvore binária. Dizemos que uma árvore binária *a1* está contida na árvore binária *a2* quando a1 é equivalente a a2 ou quando existe pelo menos uma subárvore de *a2* que seja equivalente a *a1*.

7.9.16- Desenvolva uma função recursiva que recebe entre outros parâmetros de entrada um ponteiro para a raiz de uma árvore binária. Verificar se uma árvore possui chaves repetidas.

7.9.17- Desenvolva uma função recursiva que recebe entre outros parâmetros um ponteiro para a raiz de uma árvore binária. Devolver como valor de retorno o número de átomos de um determinado nível.

7.9.18- Com base nas sequências que a seguir descrevemos, reconstitua a correspondente árvore binária: pré-ordem: A,B,C,D,E,F,G,H,I,J,K,L in-Ordem : C,E,H,G,I,J,F,K,D,L,B,A

7.9.19- Se percorrermos uma árvore na sequencia pós-ordem, pré-ordem e in-ordem, obtemos as expressões na notação pos-fixa, pré-fixa e in-fixa. Dadas as seguintes expressões:

Pré-fixa: / + - B ½ - ↑ B 2 * * 4 A C * 2 A
Pos-fixa: B – B 2 ↑ 4 A * C * - ½ + 2 A * /

7.9.20*- Desenvolva uma função que recebe como parâmetro de entrada um ponteiro para a raiz de uma árvore binária e uma determinada informação. Inserir esse átomo na árvore de tal forma que o nível das subárvore não seja superior a um. Este tipo de árvore dá-se o nome de árvore binária balanceada.

7.9.21*- Desenvolva uma função iterativa que recebe como parâmetro de entrada um ponteiro para a raiz de uma árvore binária, a chave do átomo que pretendemos remover e do seu pai. Remover esse átomo da árvore.

7.9.22- Desenvolva uma função recursiva que recebe como parâmetros de entrada um ponteiro para a raiz de uma árvore binária e uma determinada chave. Devolver como valor de retorno o nível do átomo que contêm essa chave.

7.9.23- Desenvolva uma função que recebe como parâmetro de entrada uma string com uma expressão aritmética com os operadores −, ∗ e / e os operandos identificados por apenas uma letra. Devolva por parâmetro de saída a correspondente arvore binária.

7.9.24- Desenvolva uma função que recebe como parâmetro de entrada a arvore binária montada no exercício anterior e calcula o valor da expressão que ela guarda. Suponha que existe uma rotina que dado uma letra, devolve o correspondente valor.

7.9.25- Escreva a árvore binária correspondente à seguinte express˜ao: (a + b) ∗ (c − d) − e ∗ f . Escreva os elementos da árvore considerando os percursos pré-ordem e pós-ordem.

7.9.26- O espelho T' de uma árvore binária T é definido por:
Se T é vazia, então o seu espelho é vazio; Senão T´ tem a seguinte forma: a raiz de T′ é a igual á raiz de T, a subárvore esquerda de T' é o espelho da subárvore direita de T e a subárvore direita de T′ ´´e o espelho da subárvore esquerda de T .
Desenvolva uma função recursiva que recebe um ponteiro para raiz de uma árvore binária T e a transforma no seu espelho. Devemos salientar que nenhum átomo deve ser criado e a função deve alterar apenas os ponteiros filho esquerdo e filho direito.